Tools from the Performance Evaluation Research Center (PERC)

DOE MICS PI Meeting
26 June 2002
Argonne National Laboratory

http://PERC.NERSC.GOV

Performance Evaluation Research Center

- **Help users analyze and improve application performance**
- **Understand architecture characteristics**

- **Tools**
  - —Memory Instrumentation with Sigma
  - —PAPI
  - —SvPablo
  - —Tau
  - —Rose
  - —Performance Assertions

Memory Instrumentation

- **Dynamic memory access instrumentation**
  - —collect low level memory accesses
  - —with the flexibility of dynamic instrumentation
- **Possible applications**
  - —offline performance analysis (Sigma etc.)
  - —online optimization
  - —tools to catch memory errors

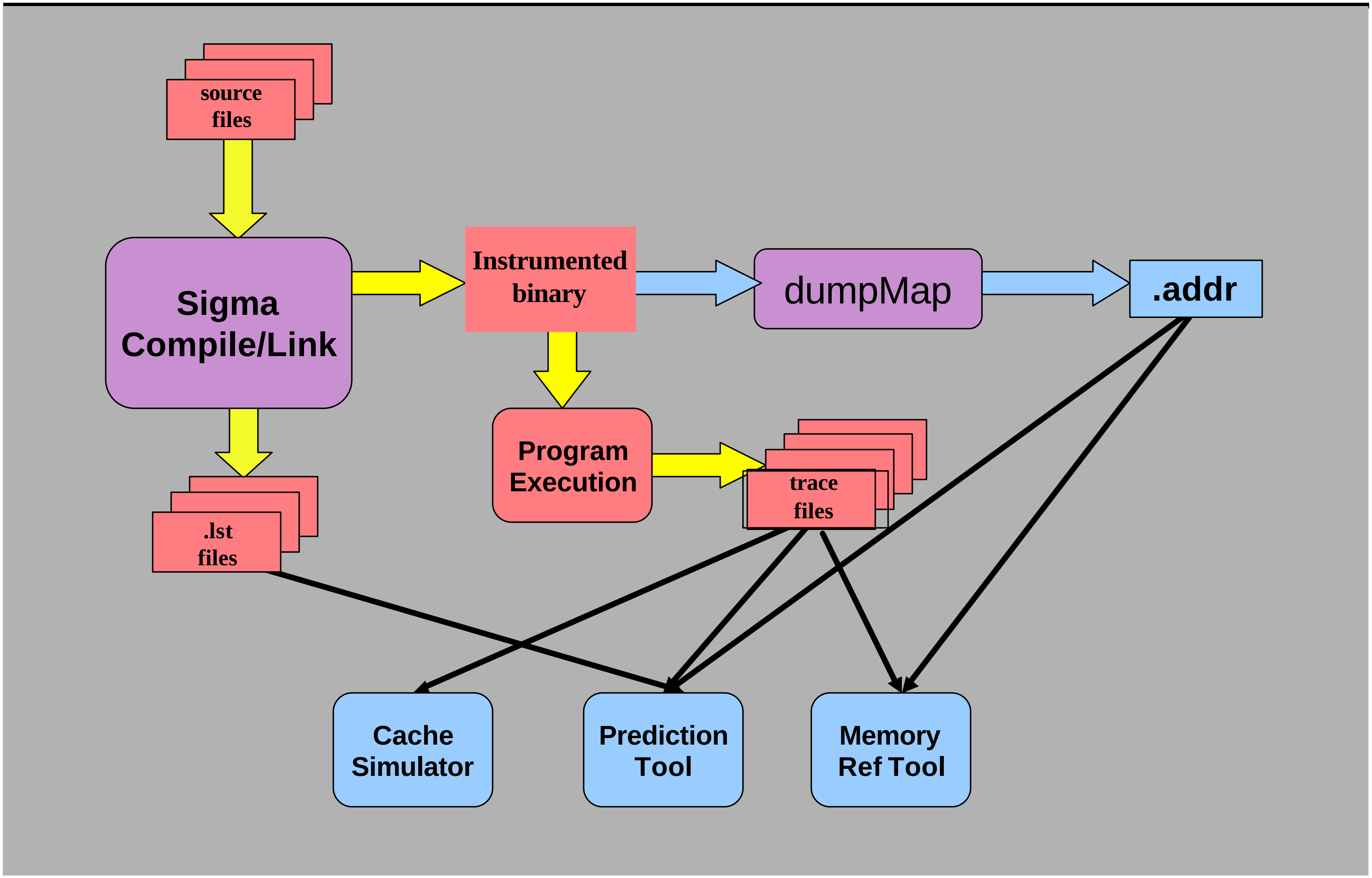
Memory Instrumentation Features

- **Finding memory access instructions**
  —loads, stores, prefetches
- **Builds on Arbitrary Instrumentation**
- **Decoded instruction information**
  —type of instruction
  —constants and registers involved in computing
    - the effective address
    - the number of bytes moved
  —available in the mutator before execution
- **Memory access snippets**
  —effective address in process space
  —byte count
  —available in mutatee at execution time

Sigma

- ✍ **Family of tools to understand caches**
- ✍ **Provide hints about restructuring**
  - —Padding (both inter and intra data structures)
  - —Blocking
- ✍ **Approach**
  - —Run instrumented program
    - Capture full information about memory use
  - —Post execution tools
    - Memory profiler
      - – share of accesses due to each data structure
    - Cache Prediction Tool
      - – Predict cache misses using symbolic equations
    - Detailed simulator
      - – Full discrete event simulator

Structure of SIGMA Data Collection



Representing Program Execution

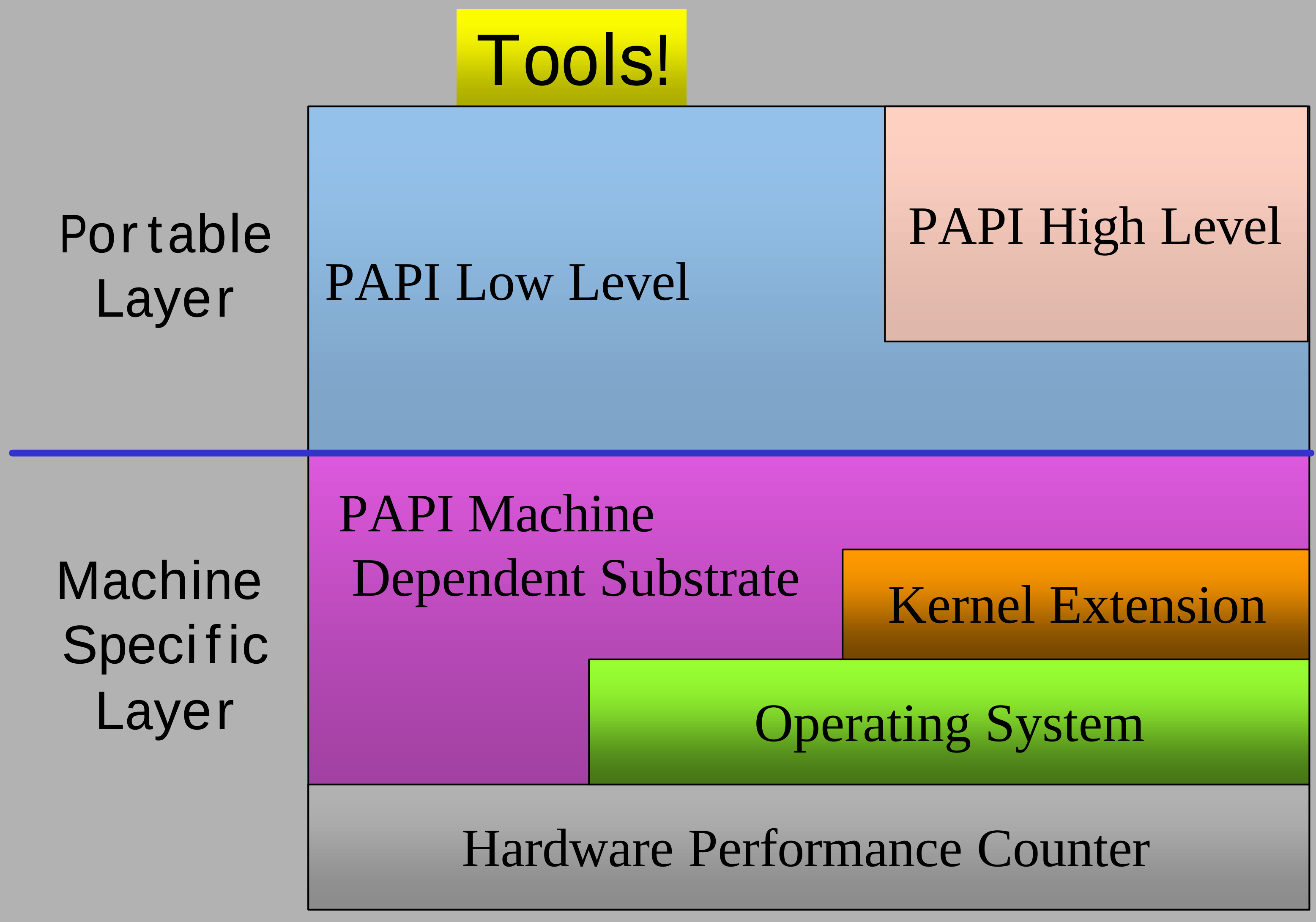
- **Capture full execution behavior**
  - —Record all basic blocks and memory addresses
  - —Produces large traces (due to looping)
- **Trace compression**
  - —Maintain pattern buffer
  - —Scan for repeating patterns
    - Extract memory strides
  - —Repeat algorithms for nested loops

	RPT	BLK1	ADR	ADR	ADR	BLK2	ADR	ADR	BLK3	
Count	250	Base	100	200	300		300	500		
Length	7	Stride	4	4	4		4	4		

PAPI

- **Performance Monitoring Hardware**
  - —Available on most modern microprocessors
  - —Consists of registers that record data about the processor's function
    - Event counts
    - Data and instruction addresses for an event
    - Pipeline or memory latencies
  - —Control registers for configuration and control
- **Performance Application Programming Interface**
  - —The purpose of the PAPI project is to design, standardize and implement a portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
  - —Parallel Tools Consortium project
    - http://www.ptools.org/

PAPI: **Implementation**



PAPI 2.1 Release

- **Platforms**
  - —Linux/x86, Windows 2000
    - Requires patch to Linux kernel, driver for Windows
  - —Linux/IA-64
  - —Sun Solaris/Ultra 2.8
  - —IBM AIX/Power3
  - —SGI IRIX/MIPS
  - —Cray T3E/Unicos
- **Fortran and C bindings and MATLAB wrappers**
- **Used in SvPablo, TAU, Vprof**
- **Planned for next release: P4, Power4, Compaq Alpha**

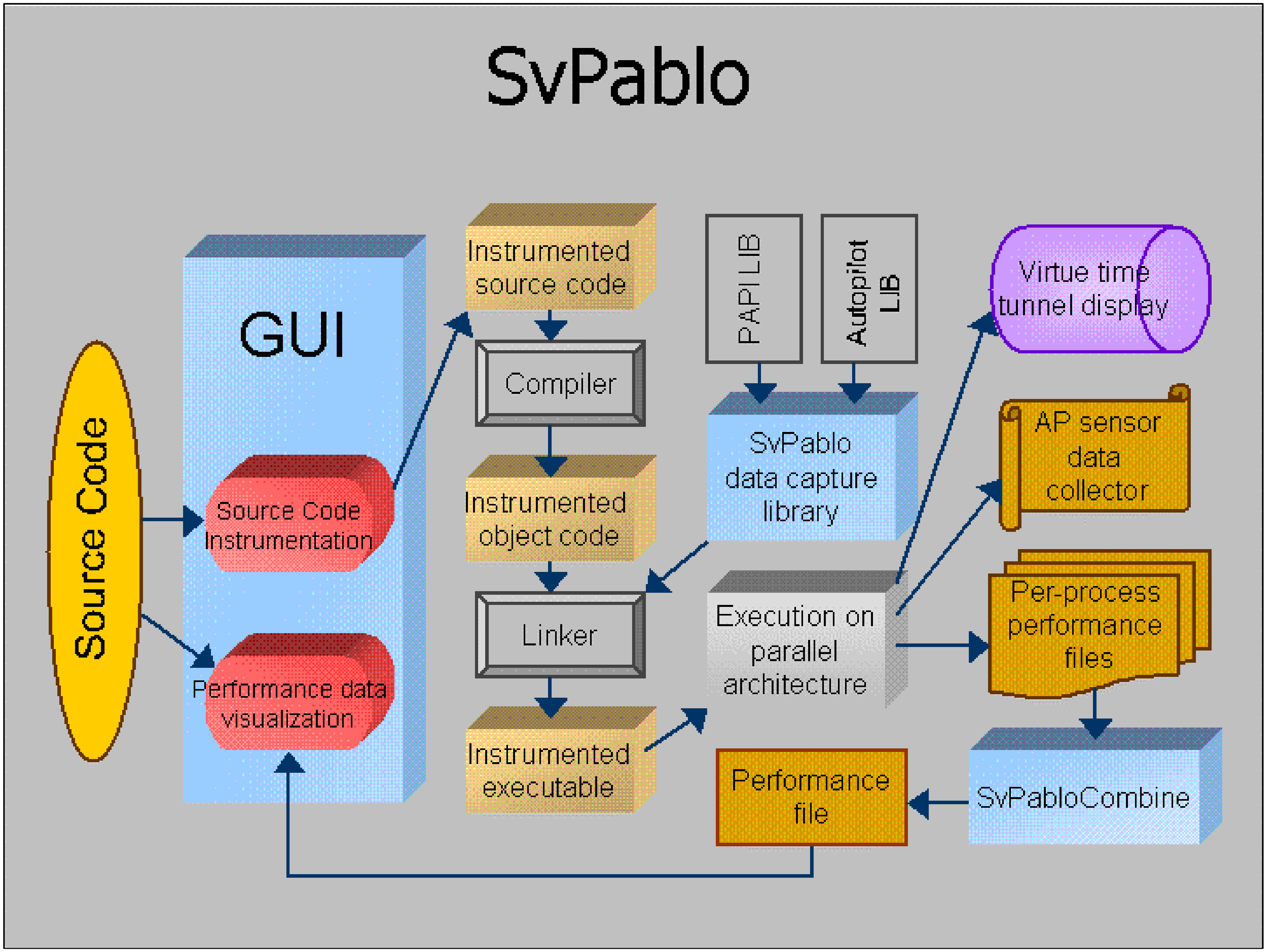
PAPI: Current Research

- Validating PAPI measurements
- Investigating tradeoffs between accuracy and efficiency of using performance monitoring hardware in counting vs. sampling modes
- Reducing PAPI overheads
- Investigating new hardware performance monitoring features (e.g., event qualification by data and instruction address, collection of latency data) using PAPI programmable events
- Dynamic instrumentation via dynaprof (uses dyninst to insert PAPI probes into executable image)

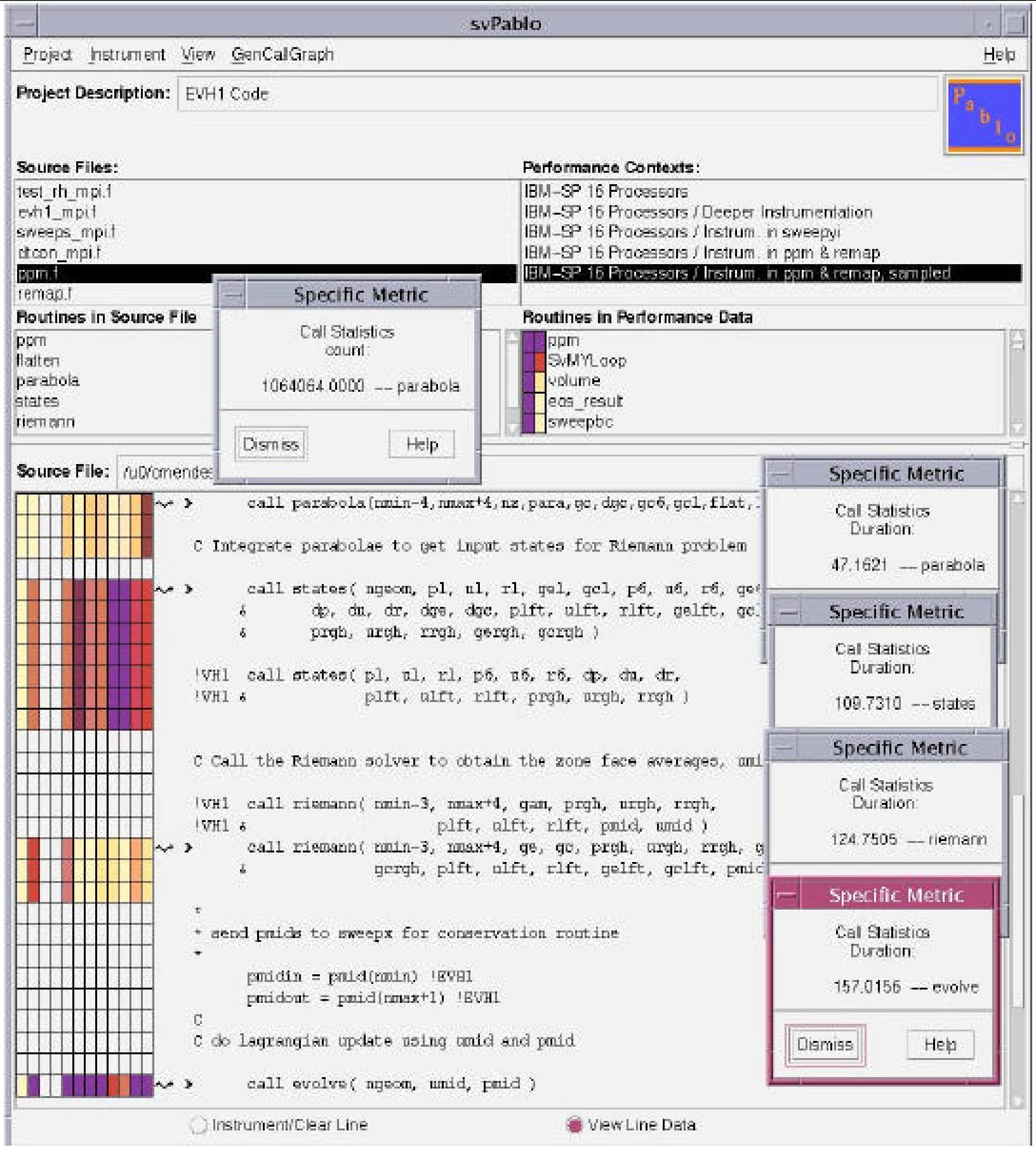
SvPablo - Main Features

- **Graphical performance analysis environment**
  - —Source code instrumentation
  - —Performance data capture, browsing and analysis
  - —F77 / F90, HPF and C language support
- **Performance capture features**
  - —software-based instrumentation (default)
  - —hardware performance counter data optional, via PAPI interface
  - —statistical summaries for long-running codes
  - —option for real-time data transmission via Autopilot sensors
- **Supported platforms**
  - —Sun Solaris, IBM SP, SGI Origin, Compaq Alpha
  - —Linux (IA-32 and IA-64)

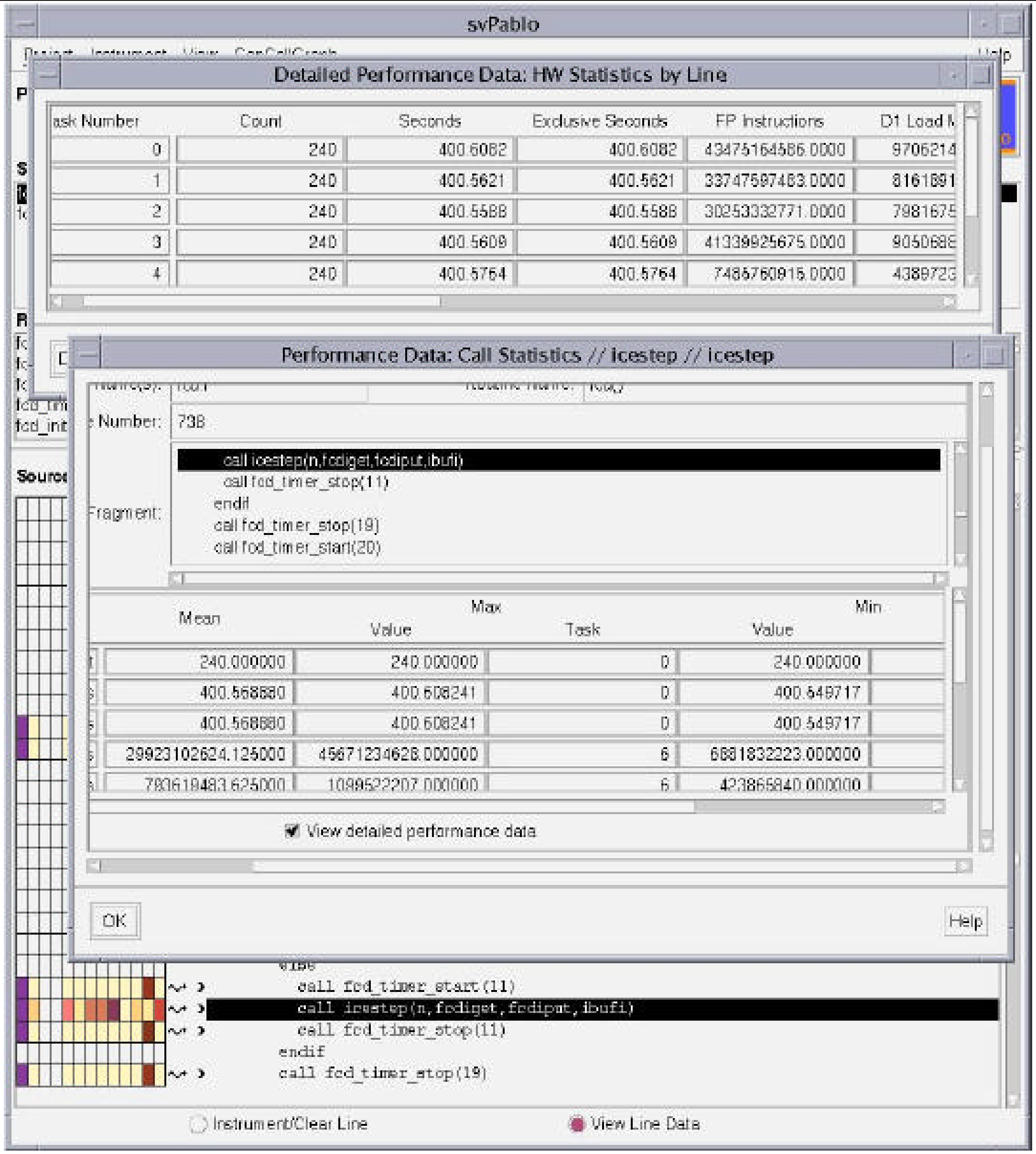
SvPablo Architecture



SvPablo GUI: Source Line Performance Data



Detailed HW Performance Data in SvPablo

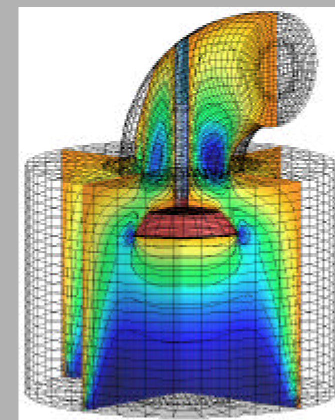


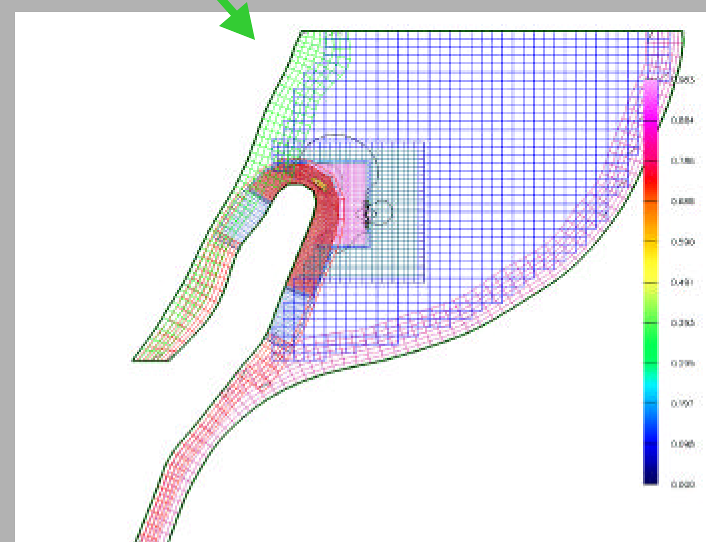
SvPablo: Status and Futures

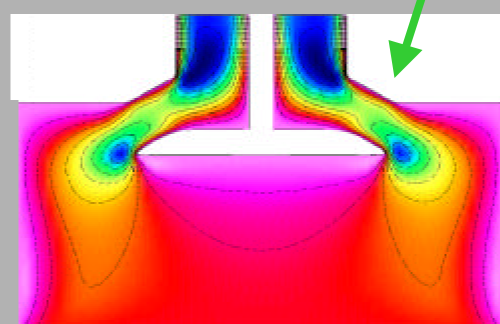
- ✍ **SvPablo extended to support new systems**
  - —Alpha and Itanium
- ✍ **Application performance analyses**
- ✍ **Design for performance model integration**
  - —comparative analysis
  - —measured and predicted behavior

ROSE Project Description

- **Goal: Simplify Scientific Software Development**
  - —Use Libraries
  - —Optimize the Libraries at Compile-Time
- **Optimize High-Level Abstractions**
- **User-Defined Abstractions Ignored By Compiler**
- **ROSE: Compiler Framework**
  - —Recognition of high-level abstractions
  - —Specification of Transformations
- **Example Problem and Results**
- **See Poster on ROSE**

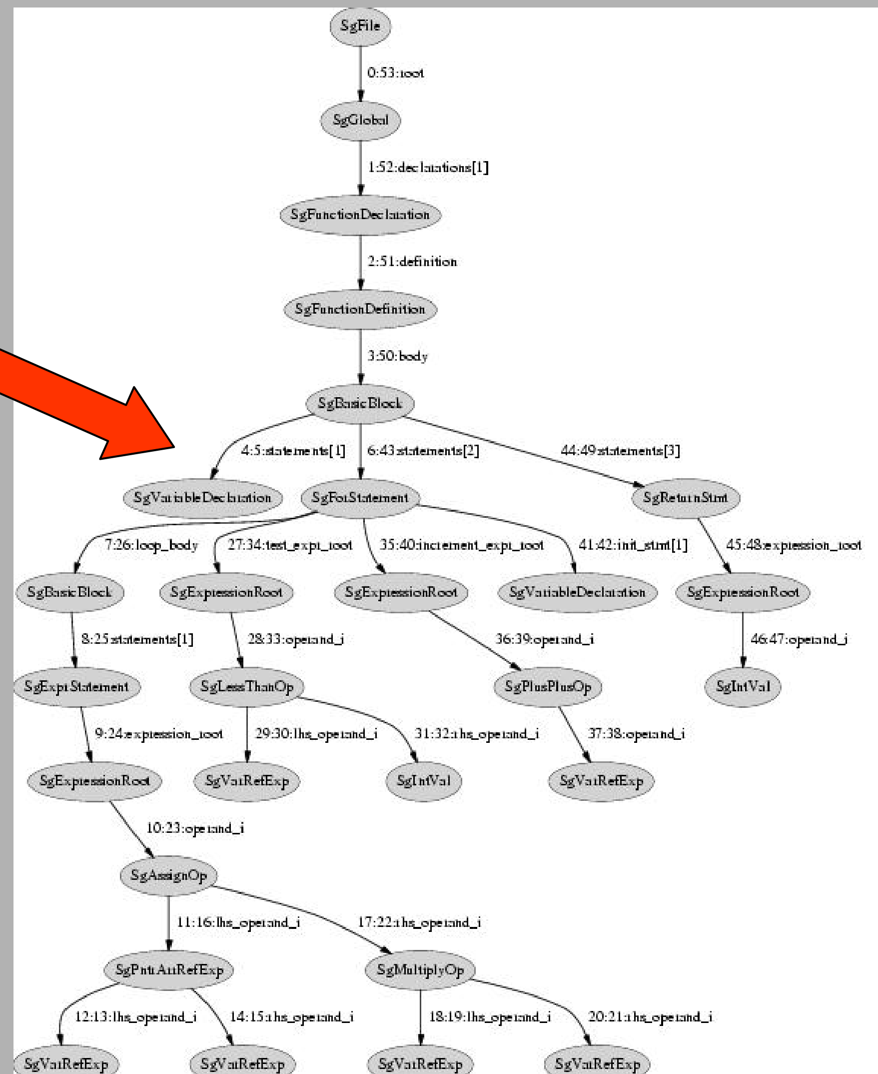






ROSE/SAGE III Abstract Syntax Tree

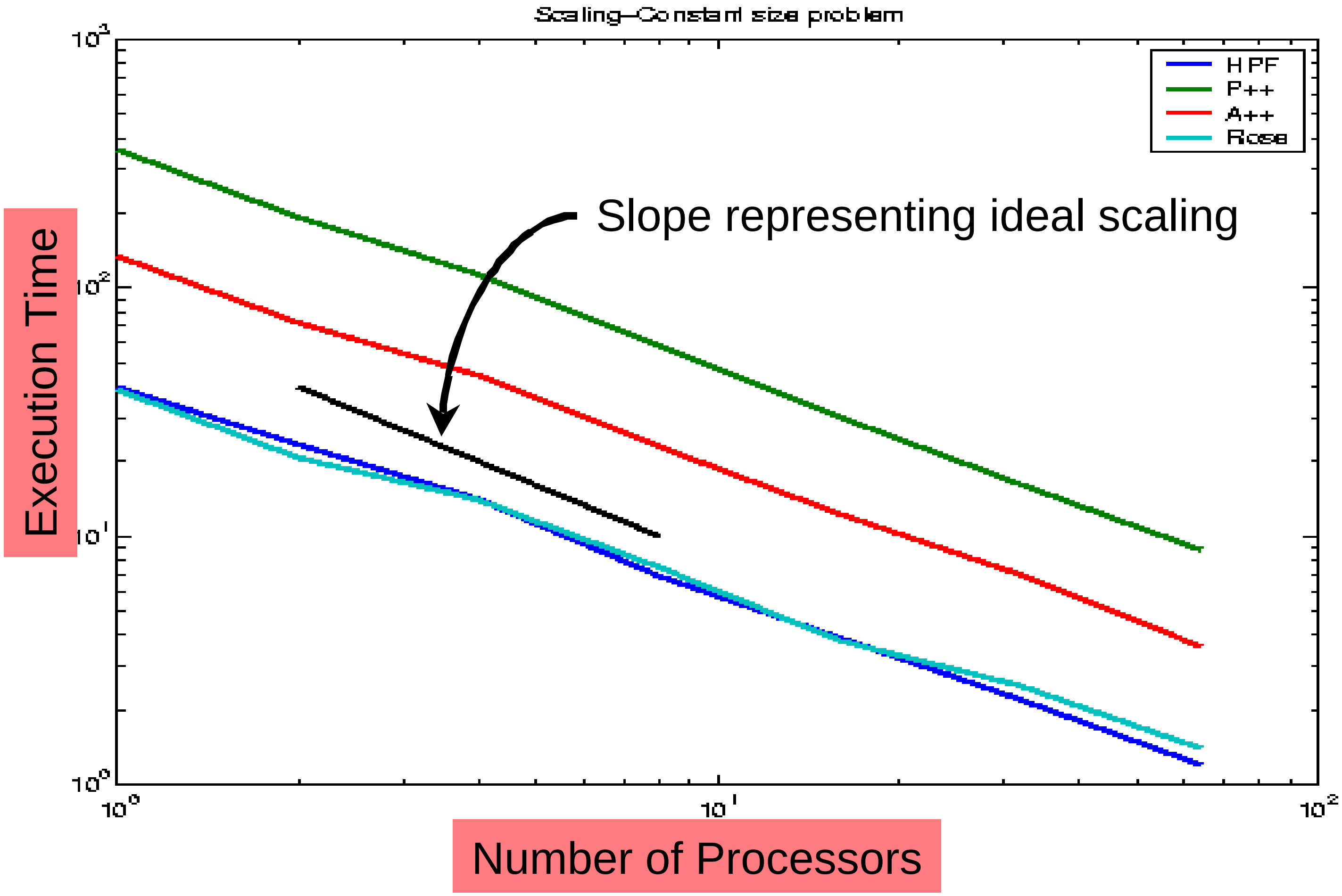
```
int main() {
  int a[10];

  for(int i=0;i<10;i++)
    a[i]=i*i;
  return 0;
}
```



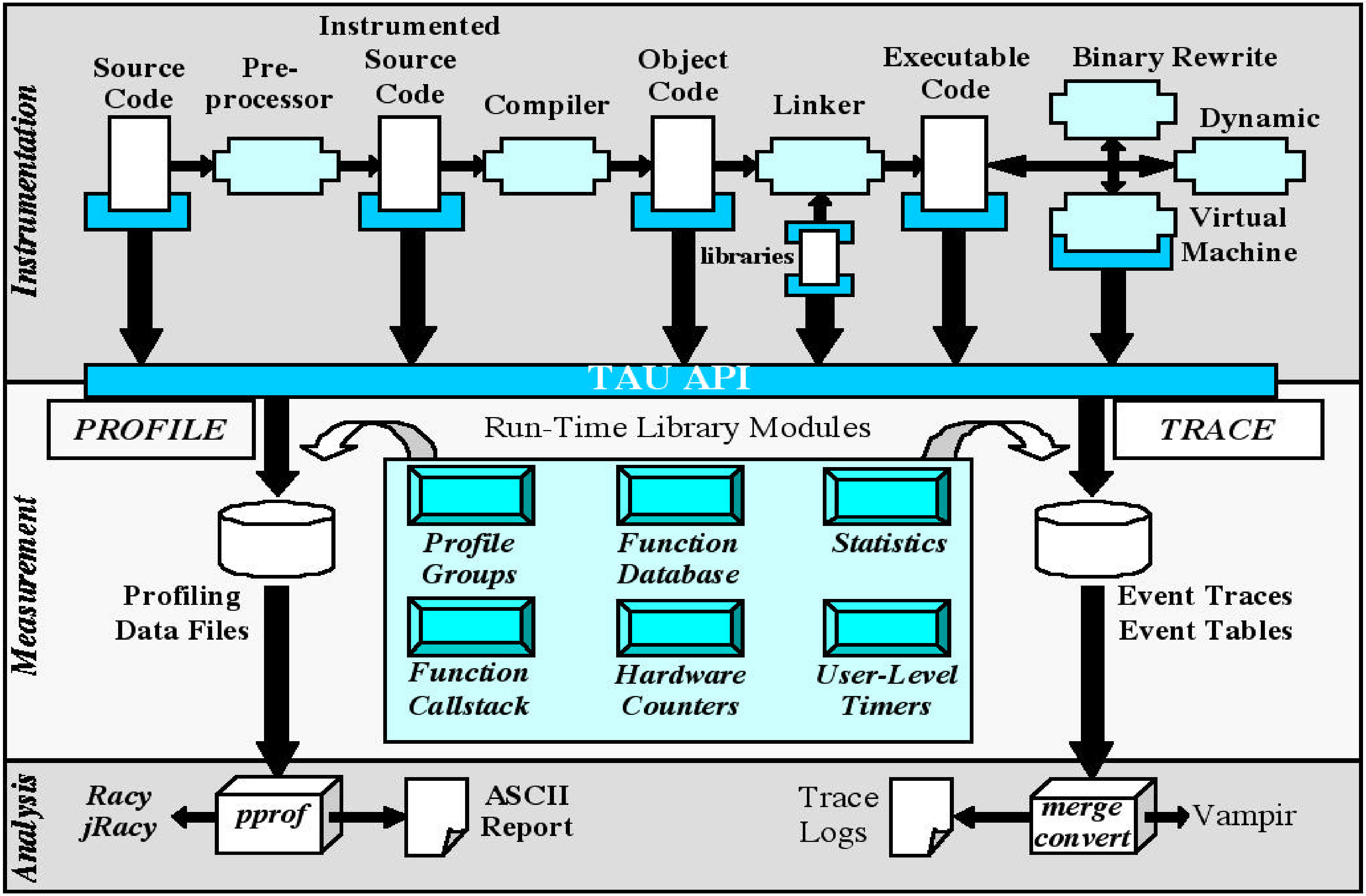
- ROSE AST Features:
  - AST Query mechanisms
  - AST Rewrite mechanisms
  - Semantic actions associated with grammar rules
  - Abstract C++ grammar is predefined
  - Higher level grammars automatically generated from library source
  - Source code generation

Relative Performance Improvement
(Using Preprocessor Build with ROSE)

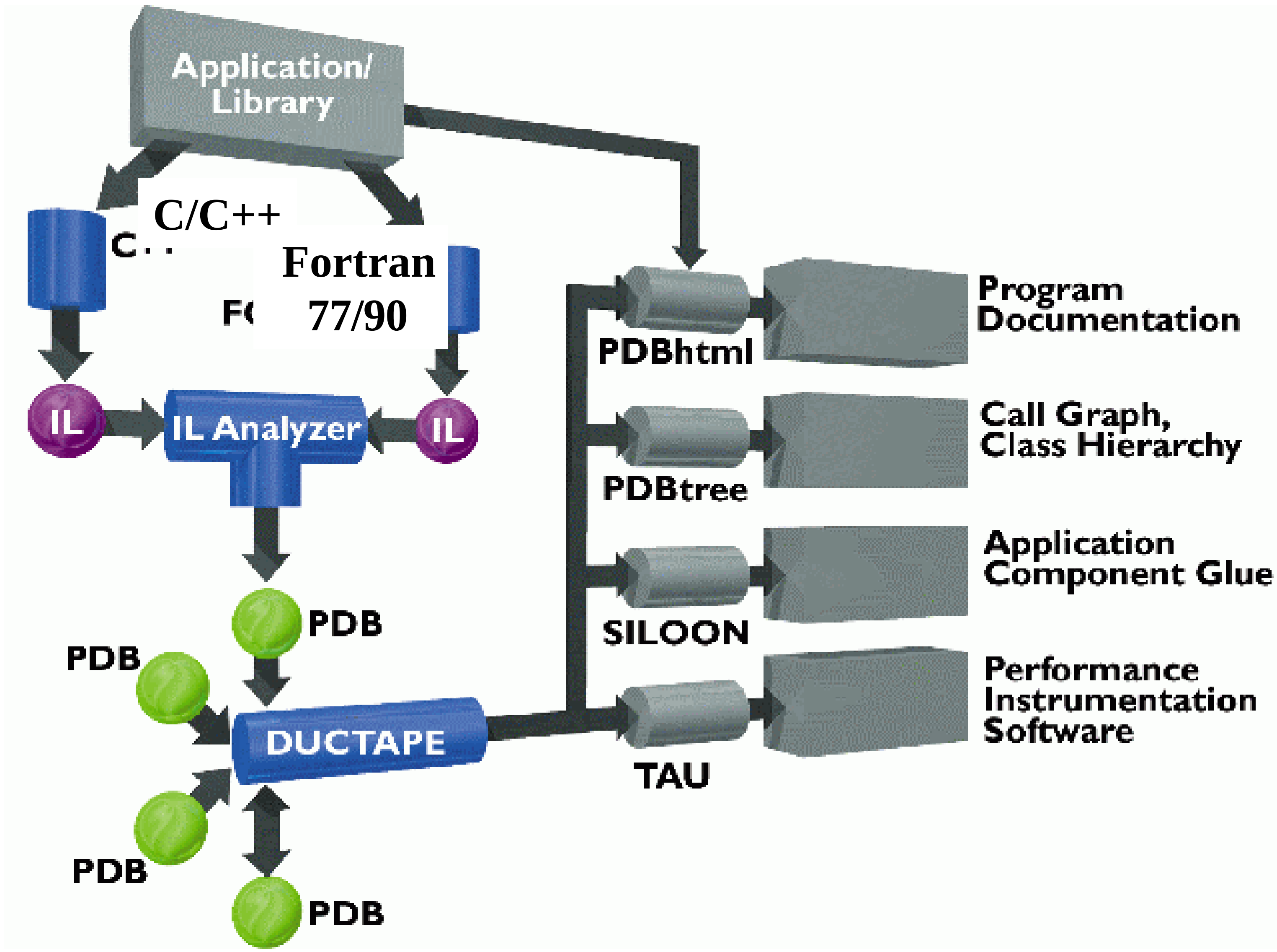
Scaling of Array Statement Abstraction
(2nd Order Linear Advection Test Problem)



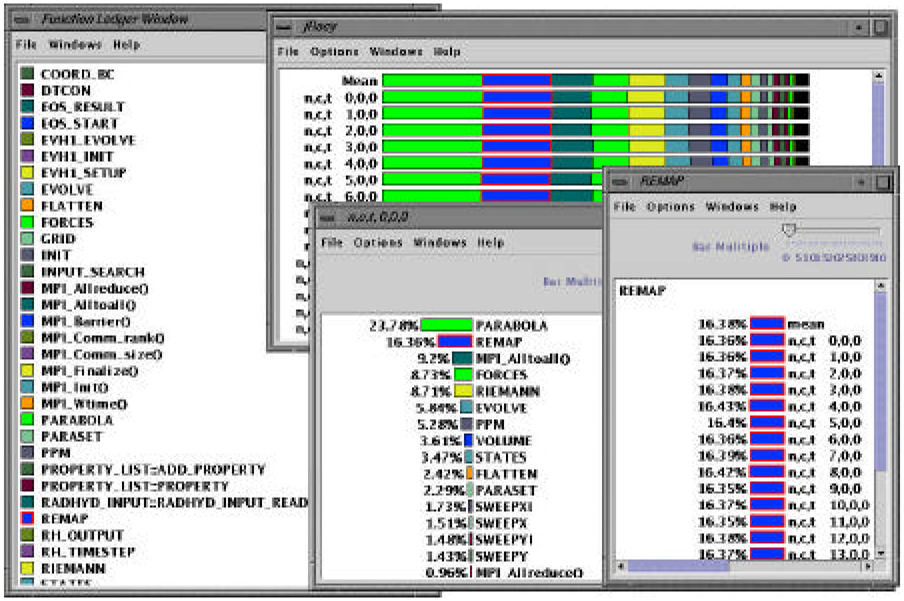
TAU: Performance System Architecture

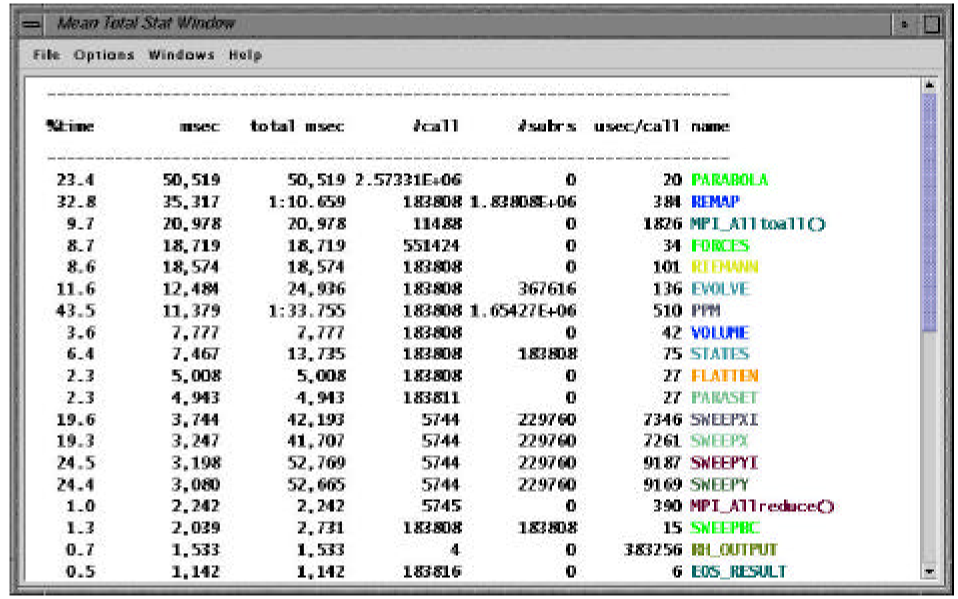


TAU: PDT Architecture and Tools

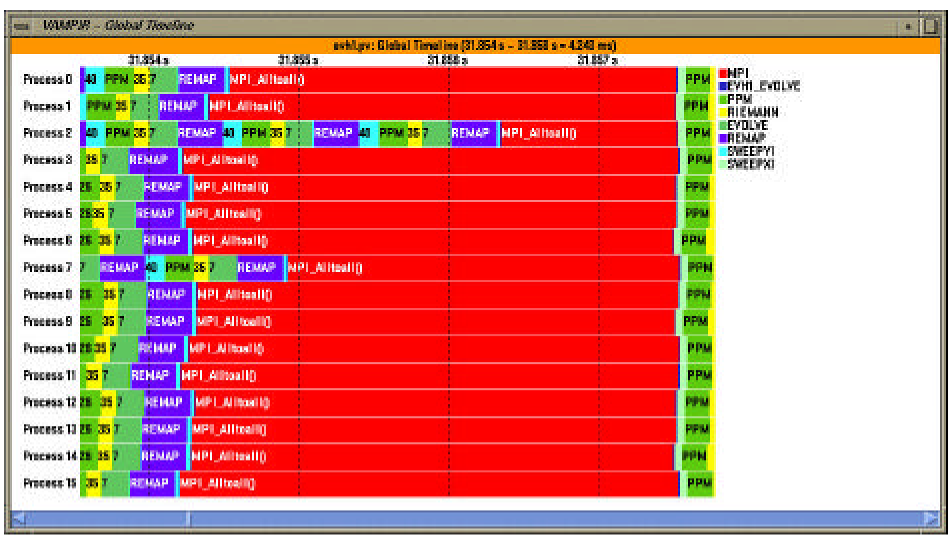


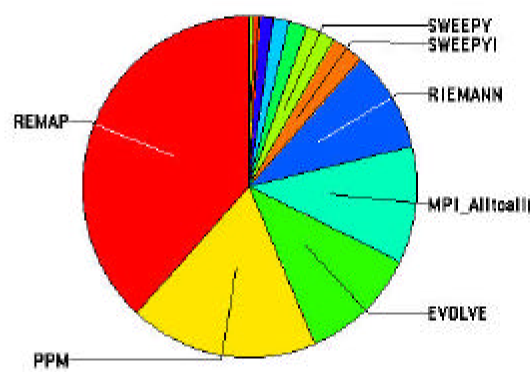
TAU: Results from EVH1





%time	msec	total msec	#call	#subrs	usec/call	name
23.4	50,519	50,519	2.57331E+06	0	20	PARABOLA
32.8	35,317	1:10.659	183808	1.83808E+06	384	REMAP
9.7	20,978	20,978	11488	0	1826	MPI_Alltoall()
8.7	18,719	18,719	551424	0	34	FORCES
8.6	18,574	18,574	183808	0	101	RIEMANN
11.6	12,484	24,936	183808	367616	136	EVOLVE
43.5	11,379	1:33.755	183808	1.65427E+06	510	PPM
3.6	7,777	7,777	183808	0	42	VOLUME
6.4	7,467	13,735	183808	183808	75	STATES
2.3	5,008	5,008	183808	0	27	FLATTEN
2.3	4,943	4,943	183811	0	27	PARASET
19.6	3,744	42,193	5744	229760	7346	SWEEPX1
19.3	3,247	41,707	5744	229760	7261	SWEEPX
24.5	3,198	52,769	5744	229760	9187	SWEEPY1
24.4	3,080	52,665	5744	229760	9169	SWEEPY
1.0	2,242	2,242	5745	0	390	MPI_Allreduce()
1.3	2,039	2,731	183808	183808	15	SWEEPBC
0.7	1,533	1,533	4	0	383256	RH_OUTPUT
0.5	1,142	1,142	183816	0	6	EOS_RESULT





Performance Assertions

- **Performance expectations are lost**
  - —When compilers introduce static decisions
  - —When users write code
    - Implicit performance expectations

- **Existing tools provide a LOT of information**
  - —Users must decide what performance data meets their expectations

- **Performance Assertions**
  - —Make explicit a developer's performance expectations for specific code segments
  - —Compare performance expectations with
    - Previous results from same/different architectures
    - Analytical comparison

- **Initial effort**
  - —Library
  - —Serial performance metrics

Performance Assertions: Goal

- Specify an equation that asserts some performance expectation
- Portable!
- Easily disabled
- Implicit notion of data collection
- Integrate application state into equation
- Forces developer to think in terms of language constructs rather than target architecture
- Assertions highlight failures, so it limits performance data glut

```
#passert ($flops/(n3*n2*n1))~1
#passert $loads == ($stores*2)
for(i3=2; i3 < n3; i3++)
  for(i2=2; i2 < n2; i2++)
    for(i1=2; i1<n1; i1++)
    {
      ...
      x(i1) = y(i1) * z(i2,i1)
    }
```

Performance Assertions: Initial Implementation

- **User Assertions**
  - —Specify an equation that asserts some performance expectation
  - —Easily disabled
  - —Implicit notion of data collection
  - —Integrate application state into equation

```
pa_start(&pa1, "$flops/(%d*%d*%d))~1", &n3, &n2, &n1);
for(i3=2; i3 < n3; i3++)
  for(i2=2; i2 < n2; i2++)
    for(i1=2; i1 < n1; i1++)
    {
      ...
      x(i1) = y(i1) * z(i2,i1);
    }
pa_end(&pa1);
```